CODE Research Corporation

SOFTWARE APPROACH TO AUTOMATIC

PATCHING OF ANALOG COMPUTER

CODE RESEARCH CORPORATION ▪ 1363 STATE COLLEGE BLVD. ▪ ANAHEIM, CALIF. 92806 ▪ 714/533-6333

FINAL REPORT

FOR

SOFTWARE APPROACH TO AUTOMATIC PATCHING OF ANALOG COMPUTER

PREPARED BY

CODE Research Corporation
1363 S. State College Blvd.
Anaheim, California 92803

UNDER

CONTRACT NUMBER NAS8-28616

November 20, 1973

TABLE OF CONTENTS

1.0 INTRODUCTION

1.1 Purpose

The Automatic Patching Verification program (APV) provides the hybrid computer programmer with a convenient method of performing a static check of the analog portion of his study. The static check insures that his program is patched as he has specified, and that the computing components being used are operating correctly.

1.2 General Method

The APV language the programmer uses to specify his conditions and interconnections is similar to the Fortran language. This similarity is, however, only in syntax; the two languages have different purposes and, therefore, their meaning is different. The APV control program reads APV source program statements from an assigned input device (normally the card reader). Each source program statement is processed immediately after it is read. A statement may select an analog console, set an analog mode, set a potentiometer or DAC, or read from the analog console and perform a test, etc. Statements are read and processed sequentially. If an error condition is detected, an output occurs on an assigned output device (normally the line printer). When a end statement is read, the test is terminated. There is no restriction on the number of statements in an APV source program.

2.0 ELEMENTS OF THE APV LANGUAGE

2.1 Constants

A constant is a string of characters representing a real number.

Examples: 3.7
-5
3872
0.7
-.382
+7.66

Only the characters 0 thru 9, +, -, and . may be used in a constant. There must be no blank characters within a constant. Constants are converted to floating point numbers in the digital computer. Therefore, a constant must lie between -10^{99} and $+10^{99}$, and there is up to 12 digits of precision available.

2.2 System Variables

A system variable is a string of characters representing the value of a particular analog device. The devices and their names are given in table 1.

Table 1. Analog Devices

	DEVICE	3 Letter Name	1 Letter Name
1.	Amplifier	AMP	A
2.	Potentiometer	POT	P
3.	Multiplier	MUL	M
4.	Resolver	RES	R
5.	Variable Function Gen.	FUN	F
6.	Trunk Line	TNK	T
7.	ADC	ADC	none
8.	Integrator	INT	I
9.	DAC	DAC	none
10.	Potentiometer Coeff.	SET	S

The general form is: NAME (N), where N is an unsigned integer between 0 and 1,000,000 written without a decimal point. N represents the number of the named device.

Examples: AMP9321) means Amplifier 321
T(67) means Trunk Line 67
DAC(0) means DAC 0

The 3 letter and 1 letter names for the same device may be used interchangeably. INT(72) means the same as I(72).

There must be no blank characters within a device name or within the integer number N.

Items 1 thru 7 in table 1 means the following: Whenever one of these device names occurs in a program the output of the corresponding device is read and the device name is replaced by the value of the ratio of the output voltage to

the reference voltage. This ratio is held as a floating point number in the digital computer.

Example: AMP(47)

When the APV program encounters this symbol, the output voltage of amplifier 47 is read and the symbol AMP(47) is replaced by the value of the ratio of this output voltage to the reference voltage.

Item 8 in table 1 means the following: Whenever an integrator name occurs in a program, this name is replaced by a value that is the equivalent voltage if it were summing its inputs. That is, this value is a measure of the integrand or initial derivative. The value is held as a floating point number in the digital computer.

Items 9 and 10 in table 1 have two different meanings, depending on how they are used in a statement. The two meanings are either:

1. Set the indicated device to a specified value.
   Example: DAC(22)
   DAC 22 is set to a specified value.
2. The device name is replaced by the last value that this device was set to.
   Example: S(14)
   This name is replaced by the last value that potentiometer 14 was set to.

2.3 Program Variables

A program variable is a string of characters representing a real number variable. It consists of from one to four

characters of which the first character must be a letter (A thru Z) and the remaining characters must be either letters (A thru Z) or digits (0 thru 9).

Examples: A
FZ
T37
L42M
KFGM
R999
W5

There must be no blank characters within a variable name. The maximum number of different program variable names used throughout an APV source program is dependent upon the size of the digital computer memory. This number is at least 250. The names of directives may not be used as variable names. Therefore, CON, MODE, ASN, VFUN, WAIT, and END must not be used as variable names.

2.4 Functions

A function is a string of characters representing the value of an arithmetic function. The general form is:

NAME (E), where E is any legitimate expression

The APV program recognizes the following functions:

NAME	Argument E	Value of Function
SINF		sine of (E x scaling constant)
COSF		cosine of (E x scaling constant)
TANF		tangent of (E x scaling constant)
ASIN	$-1 \leq E \leq +1$	(arc sine of E in radians)/scaling constant, where $-\pi/2 \leq \arcsin E \leq \pi/2$
ACOS	$-1 \leq E \leq +1$	(arc cosine of E in radians)/ scaling constant, where $0 \leq \arccos E \leq \pi$
ATAN		(arc tangent of E in radians)/ scaling constant, where $-\pi/2 < \arctan E < \pi/2$
SQRT		square root of E. If E is negative then the result will be - -E.
ABSF		absolute value of E.
COMP		comparator function. 1 if E >0 0 if E ≤0
VFUN		Variable Function Generator

The scaling constant referred to in the trigonometric functions is equal to $\pi/0.9$. This value may be easily changed by recompiling the APV program. There must be no blank characters within a function name.

Example: SQRT(+81.0)

The square root of 81.0 would be taken and then that value (9) would replace the symbol SQRT(+81.0)

3.0 EXPRESSIONS

An expression consists of a string of constants, system variables, program variables, functions, and other expressions separated by arithmetic operators.

The arithmetic operators consist of + (addition), - (subtraction), * (multiplication), and / (division). Multi-level parenthesis may be used within an expression to clarify the hierarchy of operations. In the absence of clarifying parenthesis, the hierarchy of operations follows the convention of: Multiplication and Division first, then Addition and Subtraction. Operations of equal precedence are evaluated from left to right.

Examples:
```
AX + 5.2 - 7
0.5 * KR * INT(51)
6.1/SINF(8 + A(3) )
( (A + M5)/3.6) * (1(7) + COSF(R*6) )
K - (-1(5) )
```

Evaluation of an expression results in the finding of a single real number which replaces the expression. During expression evaluation the program variables involved will take on their currently assigned values. System variables will take on the values of their respective readings from the devices involved. System variables SET(N), S(N), and DAC(N) will take on the value of their last setting.

1. Limit Option

   The value of an expression may be limited by placing limits after the expression. L and U are constants which stand for the lower and upper limits respectively. Let E by any

expression. Then: E,LIM(L,U) would mean the following:

a) if L<E<U then E would be the value of the expression.

b) if E≤L then L would be the value of the expression.

c) if E≥U then U would be the value of the expression.

U, of course, must be greater than or equal to L.

Example: 1.35, LIM(-1, +1)

The value of this expression would be equal to +1.

4.0 STATEMENTS

4.1 General

A statement is a unit of information that is processed by the APV program. It consists of a BCD card image which is broken down as follows:

column 1	:	C for comment
columns 2-5	:	not used by APV program
column 6	:	continuation for VFUN
columns 7-72	:	statement
columns 73-80	:	not used by APV program

If card column 1 is not a C character, it is not used by the APV program. Card columns that are not used by the APV program may contain any legitimate characters. Card columns 73-80 may be used for sequencing if desired.

All statements, except comment statements, must begin on or after card column 7. Blank characters may occur anywhere before or within a statement, however, they must not appear within a name or a constant. Only the VFUN statement may have continuation statements.

The card image does not necessarily have to come from a card reader. It may come from another input device such as a paper tape reader.

4.2 Comments

A comments statement has the form:

{card column 1 = C
{card columns 2-80 = comment to be output

The APV program will output the statement on the currently assigned output device and then continue.

Program Variable Definitions

A program variable definition statement equates a program variable with a value. The general form is:

Program variable name = expresssion

Examples:	KR5 = 6.3285	Defines KR5
	ZM = 2.1 * (5 -KR5)+ 0.3	Defines ZM
	J = KR5*M(4),LIM(-8,+6.3)	Defines J

The same program variable may be redefined several times throughout a program. A program variable must always be defined before it can be used.

4.3 Setting Potentiometers

A potentiometer is set by a statement of the form:

SET(N) = expression

or

S(N) = expression

where N is the potentiometer number and the expression is the value of the potentiometer coefficient to be used.

Example: SET(16) = 13.2 * I(5)/2
Potentiometer 16 is set with a coefficient equal to the computed value of the expression (13.2 * I(5)/2).

The same potentiometer may be reset several times during a program run.

4.4 Setting DAC's

A DAC is set by a statement in the form:

DAC(N) = expression

where N is the DAC number and the expression is the value that is to be output to the DAC.

Example: DAC(5) = 3.0 + 0.7 * (-R(3))
DAC 5 is set with a value that is equal to the computed value of the expression (3.0 + 0.7 * (-R(3)).

The same DAC may be reset several times during a program.

4.5 Testing

A test is performed by a statement of the form:

System variable = expression

where the system variable may have any system variable name except SET(N), S(N), or DAC(N).

When a test statement is encountered by the APV program, the following events occur. The analog device specified by the system variable is read and the expression is evaluated. If these two values lie within a certain tolerance, the test passes and the program continues. If these two values do not lie within a certain tolerance the test fails, error comment is output, and the program then continues. Specifically, if S is the ratio read, E is the value of the expression, and T is the tolerance used for testing, then the test passes if, and only if: $E - T < S < E + T$.

The tolerance used for testing may be specified in one of two ways:

1. Built-In Tolerances

   Built-in tolerances will be used unless a tolerance is specified by the programmer (see 2. below). A separate built-in tolerance exists for each arithmetic operation. They are:

Symbolic Name of Tolerance	Arithmetic Operation	Value
TOL1	+ and -	0.0001
TOL2	* and /	0.0002
TOL3	SINF and COSF	0.0005
TOL4	TANF	0.0005
TOL5	ASIN and ACOS	0.0005
TOL6	ATAN	0.0005
TOL7	ABSF	0.0002
TOL8	SQRT	0.0002
TOL9	VFUN	0.0010
TOL10	Minimum limit for reading	0.0020
TOL11	Minimum tolerance for testing	0.0001

The values of the built-in tolerances may be easily changed by recompiling the APV program.

When an expression is evaluated, the appropriate values of the built-in tolerances are added together each time an arithmetic operation is performed. For example, the expression (3 * 5 -SQRT(5)) would result in a built-in tolerance value equal to (TOL2+TOL8+TOL1). If a program variable is encountered, the previously computed tolerance for that variable will also be added into the total for the expression.

2. Specified Tolerances

A tolerance T may also be specified by the programmer using the form:

System variable = expression, TOL T

or

System variable = expression, LIM(L,U), TOL T

where T is a constant and $0.0001 \leq T \leq 1.0$.

The specified tolerance will always override the built-in tolerances.

Whichever method is used to determine the tolerance, the following will always be true. If the tolerance is ever less than TOL11, it will automatically be replaced by TOL11.

Examples:

(a) MUL(7) = 1.03/RES(1)
Multiplier 7 is read and the expression (1.03/RES(1)) is evaluated. A built-in tolerance equal to (TOL2) will be used to test the two values.

(b) POT(2) = 5.5 * POT(8) - 3.3, TOL 0.72
Potentiometer 2 is read and the expression (5.5 * POT(8) - 3.3) is evaluated. A specified tolerance of 0.72 will be used to test the two values.

(c) ADC(5) = 0.076
ADC 5 is read and compared to the value 0.076. Since a built-in tolerance of 0 is implied by the expression, the tolerance will automatically be changed to TOL11.

TOL10 is a special purpose tolerance. If the ratio of any reading from an analog device to the reference voltage is less than TOL10 in absolute value, a warning will be output and the program will continue. This enables the user to distinguish whether a wire is missing or whether the voltage is just too low to actually determine.

5.0 DIRECTIVES

5.1 CON (Console)

This directive allows the programmer to select an analog console. The form of this statement is:

CON = N

where N is an unsigned integer between 0 and 1,000. N represents the number of the analog console. There must be no blank characters within the integer N.

Once a CON statement is given, all further APV statements refer to this analog console until another CON statement which selects another analog console is encountered.

Example: CON = 3

Analog console 3 is selected for all future APV commands.

5.2 MODE

This directive allows the programmer to set an analog console to a particular mode. The form of the statement is:

MODE = code

where the first character of the code means the following:

First character of code	MODE
C	Compute
H	Hold
I	Initial Condition (or Reset)
P	Potset
S	Static Test

The remaining characters of the code may be any legitimate characters.

Examples:

(a) MODE = S

The analog computer is put in the static test mode.

(b) MODE = INITIAL CONDITION

The analog computer is put in the initial condition mode.

5.3 ASN (Assign)

This directive allows the programmer to replace variables in his program by other variables. The form of the statement is:

ASN (U_1,V_1) (U_2,V_2) (U_3,V_3) ------

where the U_i and V_i represent variable names. In all future APV statements the variable U_i will be replaced by the corresponding variable V_i before any action is taken with the variable. All future statements are affected by this, including ASN statements. The variables U_i and V_i may be either program or system variables and they may be intermixed as follows: A program variable may replace a program variable, or a system variable may replace a system variable, or a system variable may replace a program variable, or vice versa.

```
Example:  ASN (A, X3) (B, A(2))
          B = 5 * A + I(9)
```

The program variable A will be replaced by the program variable X3, and the program variable B will be replaced by the system variable A(2) in all future statements. The statement B = 5 * A + I(9) will be interpreted as the statement A(2) = 5 * X3 + I(9).

The maximum number of replacements which may be made in one run is dependent upon the size of the digital computer memory. This number is at least 20.

A variable U_i may only be replaced by one variable V_i in the same run. The following example illustrates this:

```
(ASN (A, B)
(ASN (A, C)
```

A will be replaced by B and B will be replaced by C.

5.4 TOL

This directive allows the programmer to change the values of the built-in tolerances. (Paragraph 4.3.7). The form of the statement is:

```
TOL (t, n)
```

where t is an unsigned positive integer from 1 to 11 written without a decimal point and n is a constant. t is the number of the built-in tolerance and n is the new value for that tolerance.

Built-in tolerances which are not specified by a TOL statement will be assigned their standard values as given

in paragraph 4.3.7. TOL statements may change the value of a tolerance several times throughout a program.

5.5 OUT

This directive allows the programmer to output the current values of program and system variables.

The form of the statement is:

$$OUT(V_1, V_2, V_3, \text{----}, V_n)$$

where the V_1's are either program or system variables. Program variables must have been previously defined and system variables must correspond to devices which really exist.

The OUT statement is output first and is followed immediately by a list of values which correspond to the respective variables.

Example: OUT(KAS, MUL(20), I(4))

This statement causes an output of this statement followed immediately by the current values of the program variable KAS and the system variables MUL(20) and I(4).

5.6 S

This directive allows the programmer to switch input control. The form of the statement is:

S

If current input control is located at the primary input

device, then input control is switched immediately to the secondary input device.

If current input control is located at the secondary input device, then input control is switched immediately to the primary input device.

Note that switching to a keyboard for input causes the APV source program to wait until the user has typed in a statement.

5.7 JUMP

This directive allows the programmer to pass over statements from the primary input device without executing them and go directly to a desired statement to resume execution. The form of the statement is:

```
JUMP , n
```

where n is the statement number of another statement. The jump may be in either the forward or reverse direction.

Example:

```
JUMP , 372
  .
  .
  .
  .
  .
  .
  .
  .
372     K=0.3
```

When the JUMP, 372 statement is encountered, the program goes immediately to the statement numbered 372. K=0.3 is executed next and the program continues execution from that point.

The JUMP statement automatically causes input control to go to the primary input device when the requested statement has been found.

5.8 IF

This directive allows the programmer to conditionally pass over statements from the primary input device without executing them and go directly to a desired statement to resume execution, depending upon whether the last test passed or failed. The form of the statement is:

```
IF, n
```

where n is the statement number of another statement. The jump may be in either the forward or reverse direction. If the last test passed, the IF statement does nothing and the next sequential statement after the IF statement is executed next. If the last test failed, the IF statement jumps to the statement indicated by n and execution will continue from that point. If no test statement has yet been encountered, the next sequential statement after the IF statement is executed next.

Example:

```
      I(23) = 0.5 * M (6)
      IF
   20 K = 0.2
      .
      .
      .
      .
      .
      .
      .
      .
   77 J = 0.05
```

If the test I(23) = 0.5 * M(6) passes, the program goes directly from the IF statement to statement 20. If the

test I(23)+0.5*M(6) fails, the program goes directly from the IF statement to statement 77.

The IF statement does not have to follow the test statement immediately. It could be placed any number of statements after the test statement.

The IF statement automatically causes input control to go to the primary input device under the following circumstances: (1) the IF statement jumps and, (2) the requested statement can be found.

the IF statement allows the program to attempt to diagnose a test failure.

5.9 MOVE

This directive gives the programmer the ability to move a specified number of statements away from the current statement from the primary input device. The form of the statement is:

```
MOVE, n
```

where n is a signed or unsigned integer with no imbedded blank characters. If n>0, the move is forward n statements. If n<0, the move is backward n statements. If n=0, the last statement executed is executed again. The statement MOVE without , n is automatically interpreted as MOVE , 0.

Example

```
      MOVE , 3
      X = .3
      J = 0.7
   28 M = I(6)
```

when the MOVE , 3 statement is encountered the program goes directly to statement 28 and continues execution at that point.

The MOVE statement automatically causes input control to go to the primary input device. An attempt to move backward past the first statement of the APV source program results in a move to the first statement of the program. An attempt to move forward beyond the end statement of the APV source program results in a move to the end statement of the program.

5.10 ON

This directive puts the APV control program into the pause-on mode. The format for the statement is:

ON

If a test failure occurs in the pause-on mode, input control is switched immediately to the secondary input device. If the secondary input device happens to be a 211 keyboard, a pause occurs while waiting for input.

The APV control program is automatically put in the pause-on mode if no ON or OFF statements are encountered.

5.11 OFF

This directive puts the APV control program into the pause-off mode. The format for the statement is:

OFF

If a test failure occurs in the pause-off mode, input control remains at the current input device. If the input device happens to be the INPUT file on the disc, no pause occurs after a test failure. This feature may be used to diagnose test failures.

Example: PIG = 5379 , 1

The P.I.G. is first put in the Reset mode and then a count of 5379 is set in the P.I.G. count value register and the P.I.G. oscillator is set to 1 kc.

5.12 WAIT

This directive allows the program to output a comment to the user and then wait for a response from him. It should be used when manual intervention is necessary from the user. The form of the statement is:

```
WAIT
or  WAIT , comment
```

When this statement is encountered, the following happens: First, the entire WAIT statement (card columns 1-80) is output on the currently assigned output devices. Input control is then given to the secondary input device. If this device happens to be a keyboard, a pause occurs while waiting for the input from the user. The user should take whatever manual action is requested by the wait statement and then type the "S" statement in order to continue.

Examples:
```
WAIT
WAIT, CHANGE PATCHBOARD
WAIT, FOR 1 MINUTE
```

5.13 End

An end statement has the form:

card column 1 = $
card columns 2-80 = REPEAT (optional)

This statement tells the APV control program that the APV source program run is complete. This must be the last statement of every APV source program. This statement causes an output on the currently assigned output devices. $ tells the APV control program to exit back to the operation system monitor. $ REPEAT tells the APV control program to reinitialize itself for another APV source program run.

6.0 OPERATING PROCEDURES

The APV program binary load module is on magnetic tape. It can be loaded under the modified Ames system by a $L control command on the typewriter.

There are several assigns that need to be made for the input/output devices used by APV.

Unit 1 is the primary input device normally assigned to the card reader. Unit 2 is the secondary input device normally assigned to the typewriter.

Unit 3 is the primary output device normally assigned to the line printer.

Unit 4 is the secondary or error listing device normally assigned to the typewriter or magnetic tape.

Under the modified Ames system these assignments can be made by typing a $GO command on the typewriter and reading the following control command from the card reader.

```
$WO,,122,754,201,,          1 = Reader
$WO,,123,763,202,,          2 = Typewriter
$WO,,124,745,203,,          3 = Printer
$WO,,125,763,204,,          4 = Typewriter
```

The APV source statement would follow these control cards. To start APV a $X control command is typed on the typewriter and the source statement would be read and interpreted from the card reader and listed on the line printer. If the program is operating in the pause mode each error will cause the program to request a typewriter input. In the no pause mode the program will process the source statement and list error on the error device.

7.0 PROGRAM EXAMPLE

The following figure illustrates the use of APV in performing a check on typical analog patching. The mechanization of the analog problem is shown in figure 1. No effort has been made to make this a reasonable mechanization. It has simply been constructed to use several different types of circuits. The APV program is shown in figure 2. Comments have been inserted in the program to make the source listing self-explanatory.

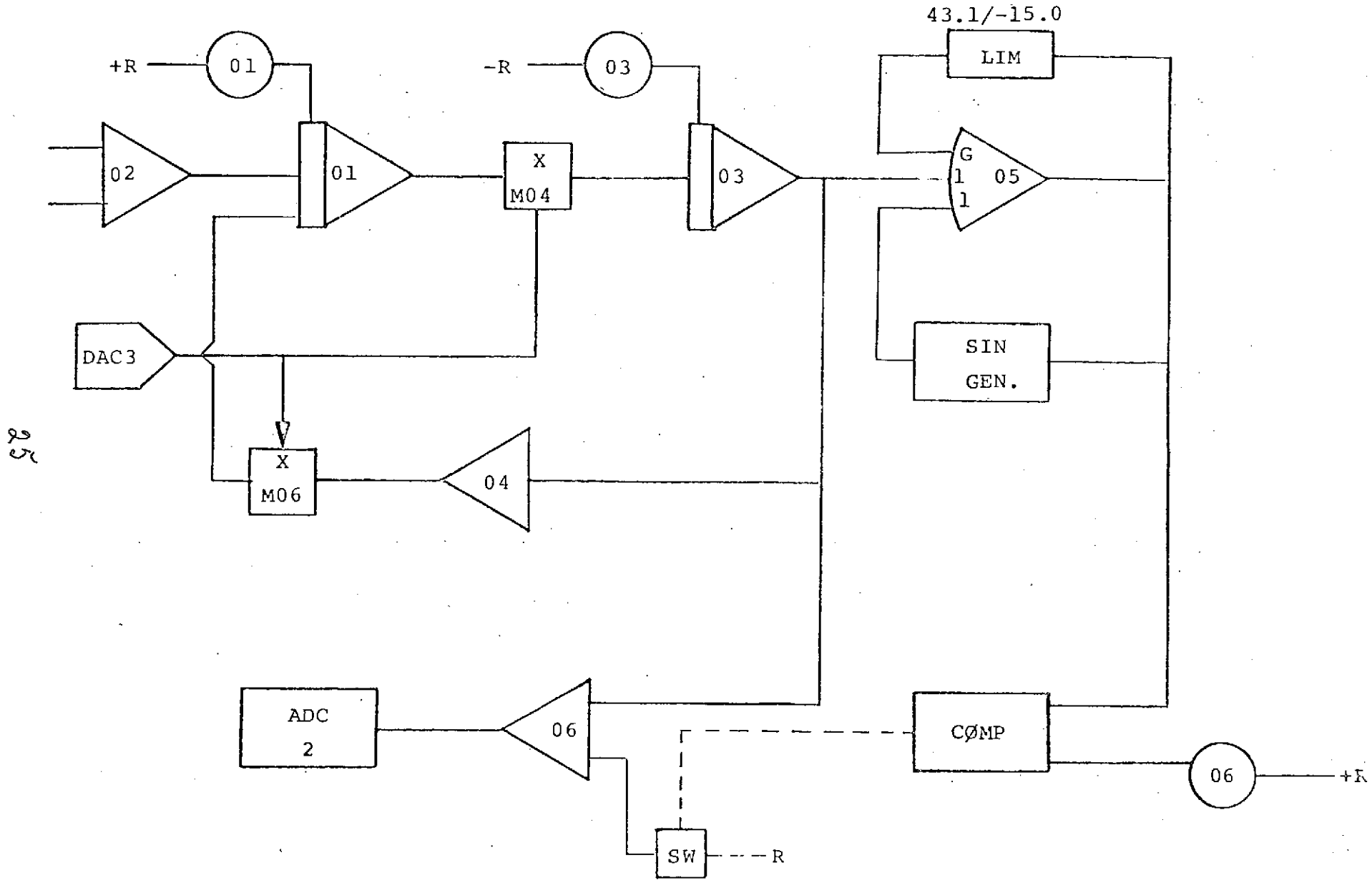

Figure 1 APV Test Circuit

FORTRAN CODING FORM

STATE NO	C	STATEMENT
		WAIT, APV PROGRAM FOR CHECKING TEST CIRCUIT
*		SELECT ANALOG CONSOLE 1
		CON = 1
*		SETUP THE POTS AND DAC
		BETA = 1.6
		GAMMA = 1.3
		SET(1) = .6147
		S(3) = 0.21105
		S(6) = .1150
		DAC(3) = BETA*GAMMA/10
*		THIS CHECKS INTEGRAND OF AMP1 FROM POT3, DAC AND AMP2
		INT(1) = S(3)*DAC(3) - AMP(2)
*		THIS CHECKS INTEGRAND OF AMP3 FROM POT1 AND DAC3
		INT(3) = SET(1)*DAC(3)
*		THIS CHECKS AMP5, THE LIMITER, AND THE SINE GENERATOR
		AMP(5) = ASIN(-AMP(3))$ LIM(-.1150$ .1431)
*		THIS CHECKS THE INPUT TO ADC2 USING LOGICAL COMP FUNC
		ADC(2) = -A(3) + COMP(POT(6)+ AMP(5))
		END